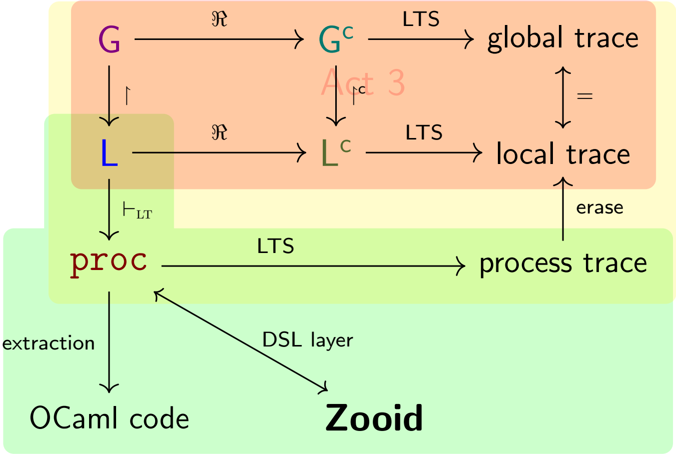
Act II

Smol-Zooid: multiparty with shallower embedding

Goals

1. Certifying **individual** processes of a **distributed system**
2. Extracting runnable code
3. Avoiding complex formalisations of binders, whenever possible

Overview


G
ℜ
G^c
LTS
global trace
Act 3
↾
↾c
=
L
ℜ
L^c
LTS
local trace
⊢LT
erase
proc
LTS
process trace
extraction
DSL layer
OCaml code
Zooid

Smol Zooid

- We combine **shallow/deep embeddings** of binders
  - Processes are defined inductively
  - Values are standard Gallina values
  - We use DeBruijn indices for the deeply embedded binders
- SZooid constructs are **well-typed by construction**

- We leverage **Coq code extraction** mechanism
- For simplicity, SZooid does not cover choices

Core Processes

In: `http://github.com/emtst/gentleAdventure`

```
Inductive proc :=
| Inact | Rec (e : proc) | Jump (X : nat)
| Send (p : participant) {T : type}
    (x : interp_type T) (k : proc)
| Recv (p : participant) {T : type}
   (k : interp_type T -> proc).
```

Payload Types

We need to define a type for payload types:

- We need a decidable equality on payload types
- We need a decidable equality on payload values

```
Inductive type := Nat | Bool | ...
Definition interp_type : type -> Type := ...
```

Semantics: overview

$$P \xrightarrow{E} P'$$

- What is an event?
- How to manage recursion?

Semantics: events

The semantics is an LTS:

- the labels are the **communication events**
- it is parameterised by a **payload interpretation function**
- traces are obtained as the greatest fixpoint of the LTS step

```
Inductive action := a_send | a_recv.
Record event interp_payload :=
  { action_type  : action;
    subj         : participant;
    party        : participant;
    payload_type : type;
    payload      : interp_payload payload_type }.
```

Semantics: Recursion Variables

`p_unroll` exposes the first communication action in a process (unfold recursion):

```
Definition p_unroll e :=
match e with
| Rec e' => p_subst 0 e e'
| e' => e'
end.
```

Semantics: Recursion Variables

```
Fixpoint p_subst d e' e :=
match e with
| Rec e => Rec (p_subst d.+1 e' e)
| Jump X => if X == d then p_shift d 0 e' else e
| ...
end.

Example ex_p_subst:
p_subst 0 (ping_Alice) (Rec (Jump 1)) = Rec ping_Alice.
```

Semantics: recursion unrolling

```
(* unroll uT. Alice!0. T to Alice!0. uT. Alice!0. T *)
Example ex_p_unroll:
p_unroll (Rec (@Send Alice Nat 0 (Jump 0)))
= @Send Alice Nat 0 (Rec (@Send Alice Nat 0 (Jump 0))).
```

Semantics: step

The step of the LTS is defined as a **function**:

```
Definition step' e E :=
  match e with
  | Send p T x k =>
    if (action_type E == a_send) && (party E == p) &&
       (eq_payload (payload E) x)
    then Some k else None
  | Recv p T k => ...   | _ => None
  end.
Definition step e := step' (p_unroll e).
```

Semantics: step

```
Definition event_alice: event interp_type :=
{| action_type := a_send;
   from := Bob;
   to := Alice;
   payload_type := Nat;
   payload := 0 |}.
Example ex_step: step infinite_ping_Alice event_alice
                = Some infinite_ping_Alice.
```

Local Types

- Local types for processes
- Notion of "Being well-typed"
- Simultaneous construction of processes & well-typeness proof

Local Types

```
Inductive lty :=
  | l_end
  | l_jump (X : nat)
  | l_rec (k : lty)
  | l_send (p : participant) (T : type) (l : lty)
  | l_recv (p : participant) (T : type) (l : lty).
```

Type System

```
Inductive of_lty : proc -> lty -> Prop :=
| lt_Send    p T k L x :
    of_lty k L -> of_lty (@Send p T x k) (l_send p T L)
| ...
.

Example ex_of_lty:
of_lty infinite_ping_Alice
(l_rec (l_send Alice Nat (l_jump 0))).
```

Smol Zooid: Smart Constructors

- It would be tedious to type up both a local type and a process
- Users would need to provide a proof that processes are well-typed

We define **SZooid** (Smol Zooid), to write well-typed processes by construction, avoiding repetition.

Smol Zooid: Smart Constructors

```
Definition SZooid L := { p | of_lty p L}.

Definition z_Send  p T x L (k : SZooid L)
  : SZooid (l_send p T L)
  := exist _ _ (lt_Send p x (proj2_sig k)).
...
```

Smol Zooid: Smart Constructors

```
Example Alice_smart_constructor :=
        z_Rec (z_Send Bob Nat 42 (z_Jump 0)).
(*
Alice_smart_constructor:
        SZooid (l_rec (l_send Bob Nat (l_jump 0)))
*)
```

Conclusion

1. Define processes and local types
2. Semantics of processes
3. Automatic construction of local types

- We also have *code extraction*
- and *subject reduction*